CSE 326
Lecture 21

Spring Quarter 1997

Radix Sort

- Perhaps the first mechanical sorting algorithm ever used. Used in the 1890 U.S. census by Hollerith.
- Does not use comparisons like quicksort, mergesort and relatives
- Very fast O(n) time for fixed length objects.
- Radix parameter should be chosen well to optimize performance

Example (Pass 1)

- 6 bit numbers with a radix of 2

101101
001100
111001
111000
000101
010010
101111
110100
100111
001101
101011
100001

taverse up

bucket 00

bucket 01

bucket 10

bucket 11

Example (Pass 2)

001100
111000
110100
101101
111001
000101
001101
100001
010010
101111
100111
101011

bucket 00

bucket 01

bucket 10

bucket 11

Example (Pass 3)

100001
010010
110100
000101
100111
111000
111001
101011
001100
101101
001101
101111

bucket 00

bucket 01

bucket 10

bucket 11

Radix Sort Parameters

- n keys
- b bit integers - how to choose r

  - r divides b
  - $n/2^r$ is not too small
  - 2^r not excessively large

  Example: 1 million 64 bit integers, r = 16 is an excellent choice
  1 million / 2^{16} is approximately 15
- w length alphabetic strings - here we are in base 26

  typical choices are 1, 2, 3, or 4,
  number of buckets is 26^2 = 676, 26^3 = 17,576, 26^4 = 456,976
- variable length alphabetic strings can be done

  1. Sort all strings by length
  2. Radix sort longest strings first, include shorter strings in the pass where the characters in the shorter strings become significant.

Analysis of Radix Sort

- Integers case only:

  b/r passes

  in each pass n keys are moved and 2^r buckets are traversed

  Total time = $O(b/r\ (n + 2^r))$

  Since b and r are constants then total time = O(n). Linear time!

Alternative Implementations of Radix Sort

- Linked list implementation

  Array of size n plus array of pointers of size 2^r (buckets)

  For b/r passes scan the array right to left inserting the keys into the approriate bucket. Scan the buckets left to right inserting the keys back into the main array.
- Array implementation

  Two arrays of size n plus a count array of size 2^r

  b/r passes:

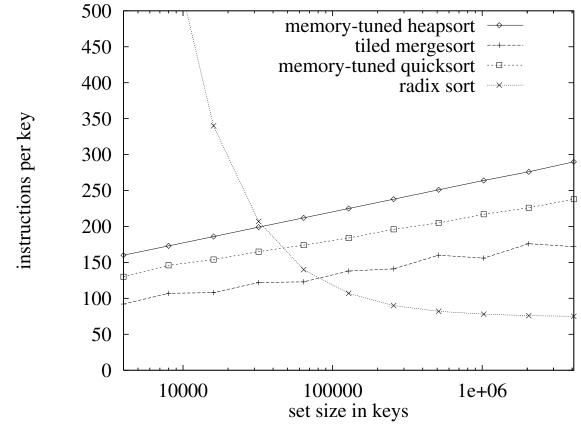
  1. Scan the source array counting the number of keys destined for each bucket.
  2. Compute the offsets for the buckets from the counts.
  3. Scan the source array inserting the keys in the destination array. With each key inserted decrement the appropriate offset. (reverse role of source and destination for next pass)

Example of Array Algorithm

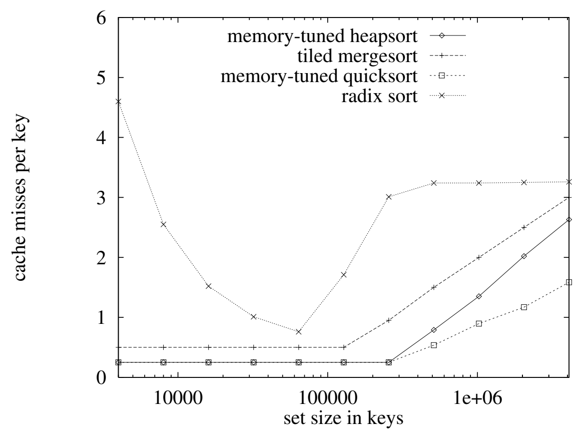
Source		Destination
101101	1	
001100	2	
111001	3	
111000	4	
000101	5	
010010	6	
101111	7	
110100	8	
100111	9	
001101	10	
101011	11	
100001	12	

Counts	Offsets
00	00
01	01
10	10
11	11

Sorting Instruction Count



Sorting Cache Misses



Sorting Execution Time